

ENUNCIAT PAC 1

Programació reactiva: Formularis reactius

Desenvolupament front-end avançat

Màster Universitari en Desenvolupament de llocs i aplicacions web

UOC

Universitat Oberta
de Catalunya

Contingut

- Format d'entrega
- Configuració entorn
- Enunciat
- Puntuació



Format d'entrega

Es lliurarà un fitxer **zip** amb:

- El projecte **Angular** proporcionat al campus **blog-uoc-project-front** comprimit en un **zip** sense incloure el directori "**node_modules**" amb la implementació de la solució.
- Un document de text (si fos necessari) comentant qualsevol aspecte que sigui rellevant a tenir en compte, per exemple, algunes decisions de disseny, alguna part que no funciona o ha quedat pendent, ... En definitiva, aspectes que jo com a "corrector" hagi de tenir en compte.

Configuració entorn

El primer que haurem de fer és veure el vídeo del campus **1-ExplicaciónFront**.

En aquest vídeo s'explica com hauria de quedar l'aplicació una vegada acabada. És a dir, es mostra el resultat que heu de ser capaços d'implementar vosaltres.

Seguidament, podeu visualitzar els vídeos **2-ExplicaciónPostman** i **3-Swagger** on comento com utilitzar tant el **Postman** com el **Swagger**. No obstant això, per utilitzar aquests dos programes primer hem de tenir l'API funcionant. Per tant, comencem a comentar com configurar i aixecar l'API. Aquí cal afegir que, si estudiem en el seu moment l'aplicació **MessageApp** de la teoria del tema 1, ens resultarà molt senzill tenir l'API d'aquesta pràctica 1 funcionant.

Bàsicament per tenir l'API funcionant hauríem de:

- Crear una base de dades amb el **pgAdmin** que es digui **blog-uoc-project**
- Baixar el fitxer **4-blog-uoc-project-api.zip**, descomprimir-lo i executar les comandes:

o **npm install** (instal·lar tots els paquets)

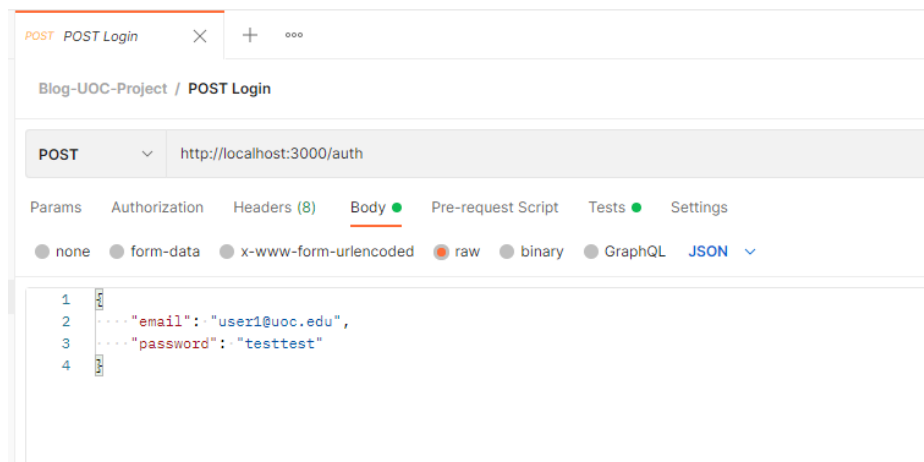
o **nest start --watch** (iniciar l'API, la primera vegada ens crearà totes les taules a la base de dades, **recordeu que primer l'hem de tenir creada**)

Un cop la base de dades creada i l'API aixecada, podem configurar el **Postman**.

Podeu tornar a veure el vídeo **2-ExplicaciónPostman** per recordar la configuració de l'autenticació.

A continuació, us mostro una captura de pantalla de cada crida que haureu d'utilitzar com a ajuda:

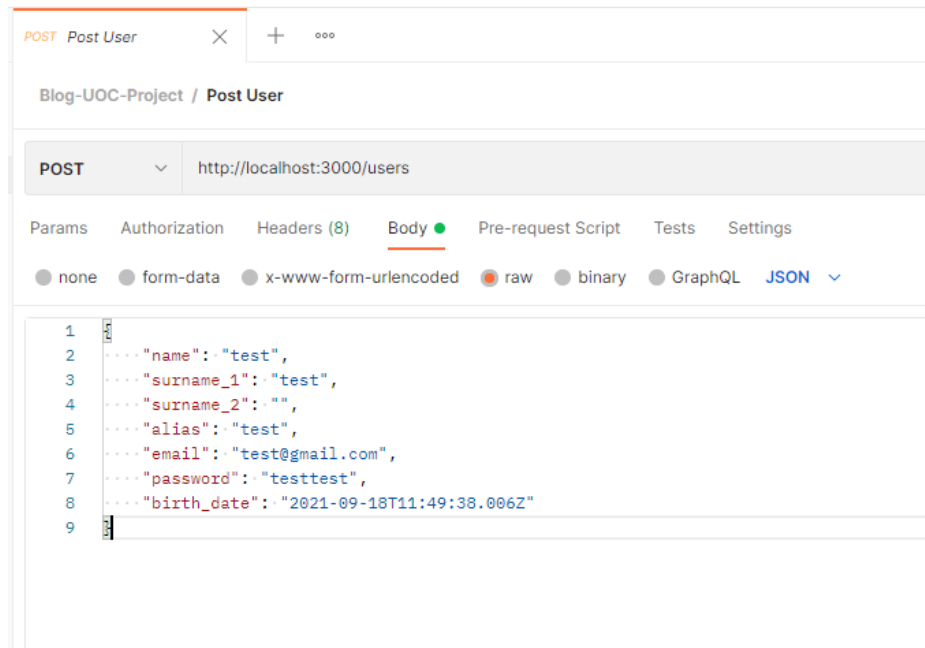
- Autenticació a la plataforma:



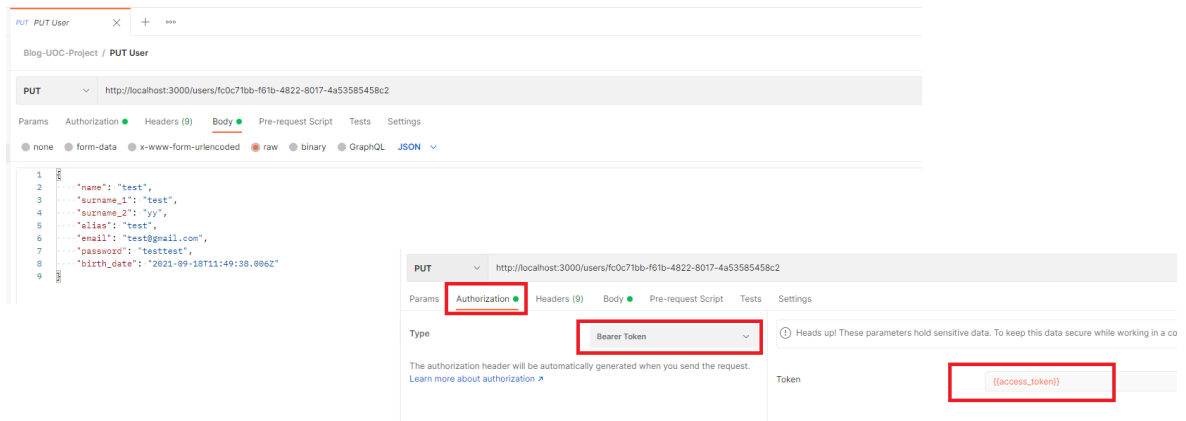
Recordeu afegir les següents línies a la pestanya **Tests**:

```
var bodyAuth = JSON.parse(responseBody)
pm.environment.set("access_token",bodyAuth.access_token);
```

- Registre d'un usuari nou:

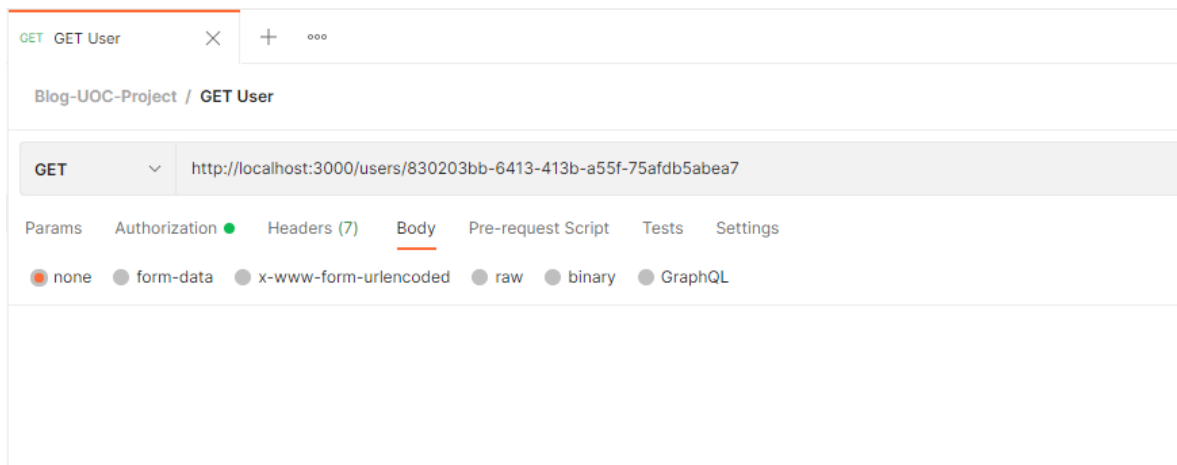


- Actualitzar les dades d'un usuari:

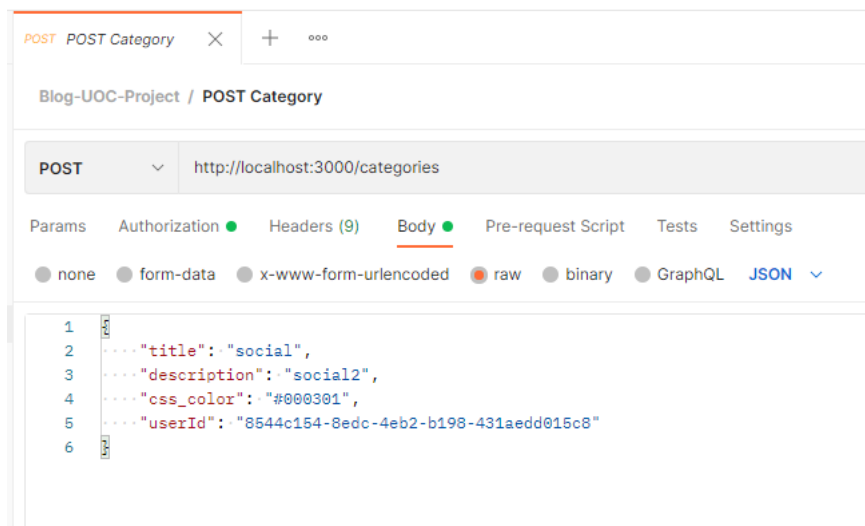


Recordeu que totes les crides que necessitin autenticació han de tenir la configuració **Bearer Token** a la pestanya **Authorization** i el token com `{{access_token}}`.

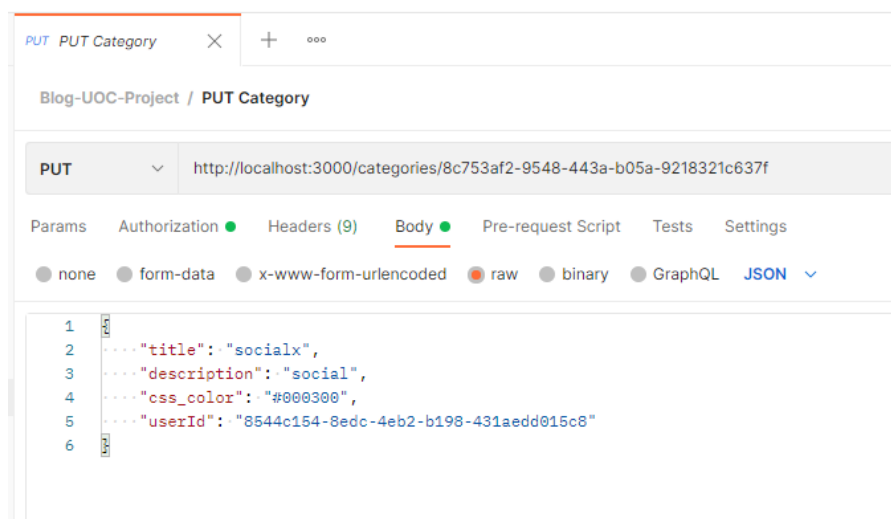
- Retornar dades d'un usuari segons el seu **id**:



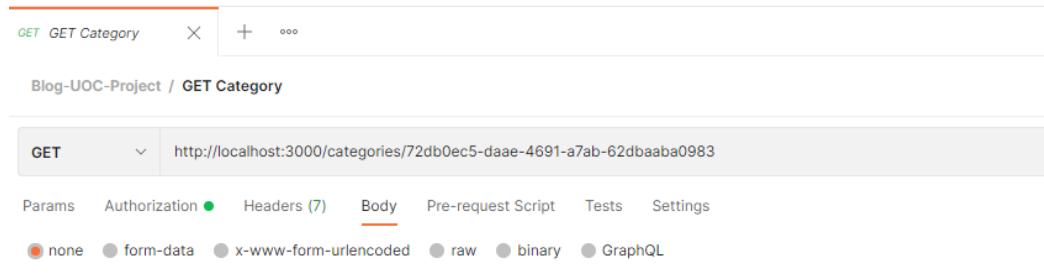
- Alta de categoria:



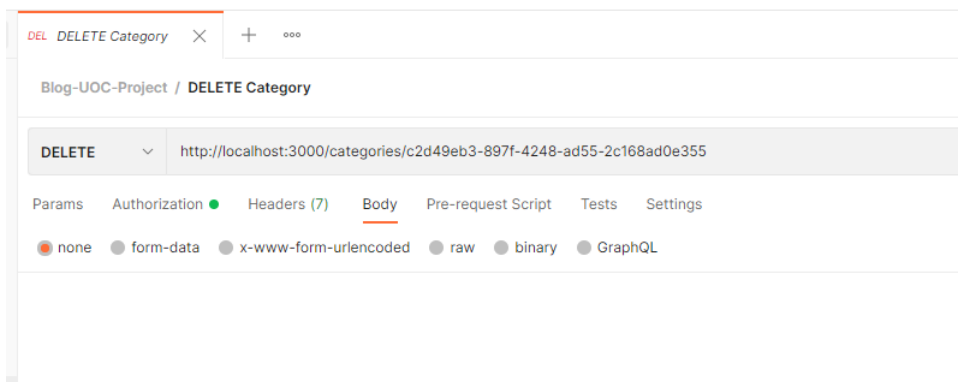
- Actualitzar una categoria:



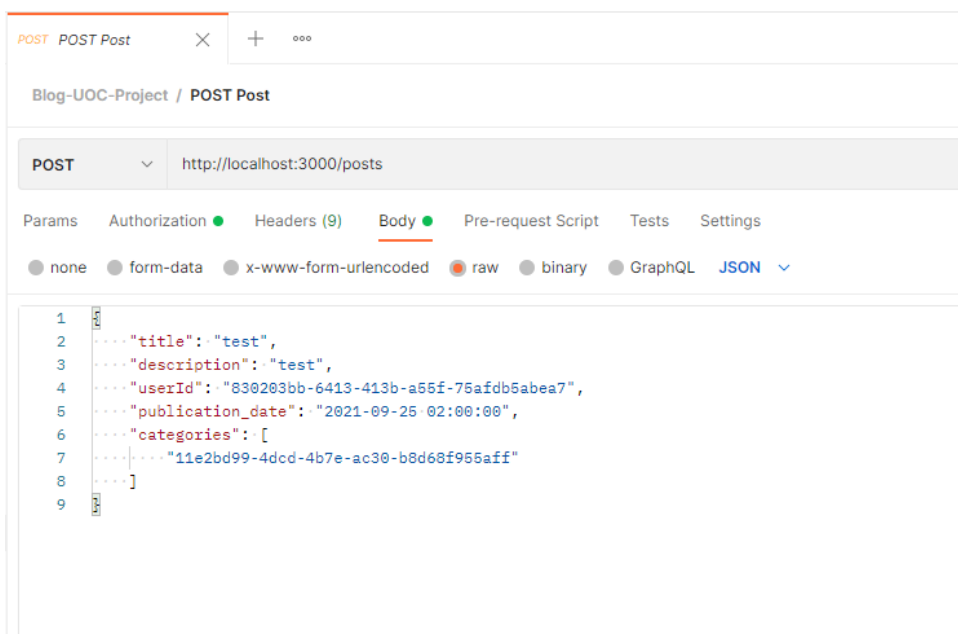
- Retornar una categoria segons el seu id:



- Eliminar una categoria:



- Donar d'alta un post, fixeu-vos que hem de passar-li un array amb els IDs de les categories associades:



- Edició de un post, igual que abans, fixeuvos que passen el array de **ids** de les categories associades:

PUT PUT Post

Blog-UOC-Project / PUT Post

PUT http://localhost:3000/posts/6dd2600b-2549-4954-aa79-a5020cdcfad6

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1  {
2    "title": "first post si",
3    "description": "tratra si",
4    "num_likes": 12,
5    "num_dislikes": 3,
6    "userId": "ced861cf-219b-43dc-a17d-d5b83445de16",
7    "categories": [
8      "c04f6300-f221-4667-b13d-94c68e14e050",
9      "41936570-352b-4f81-a9e3-712635b60ddb"
10   ]
11 }
12

```

- Retornar tots els posts:

GET GET Posts

Blog-UOC-Project / GET Posts

GET http://localhost:3000/posts

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY
Key

Body Cookies Headers (8) Test Results

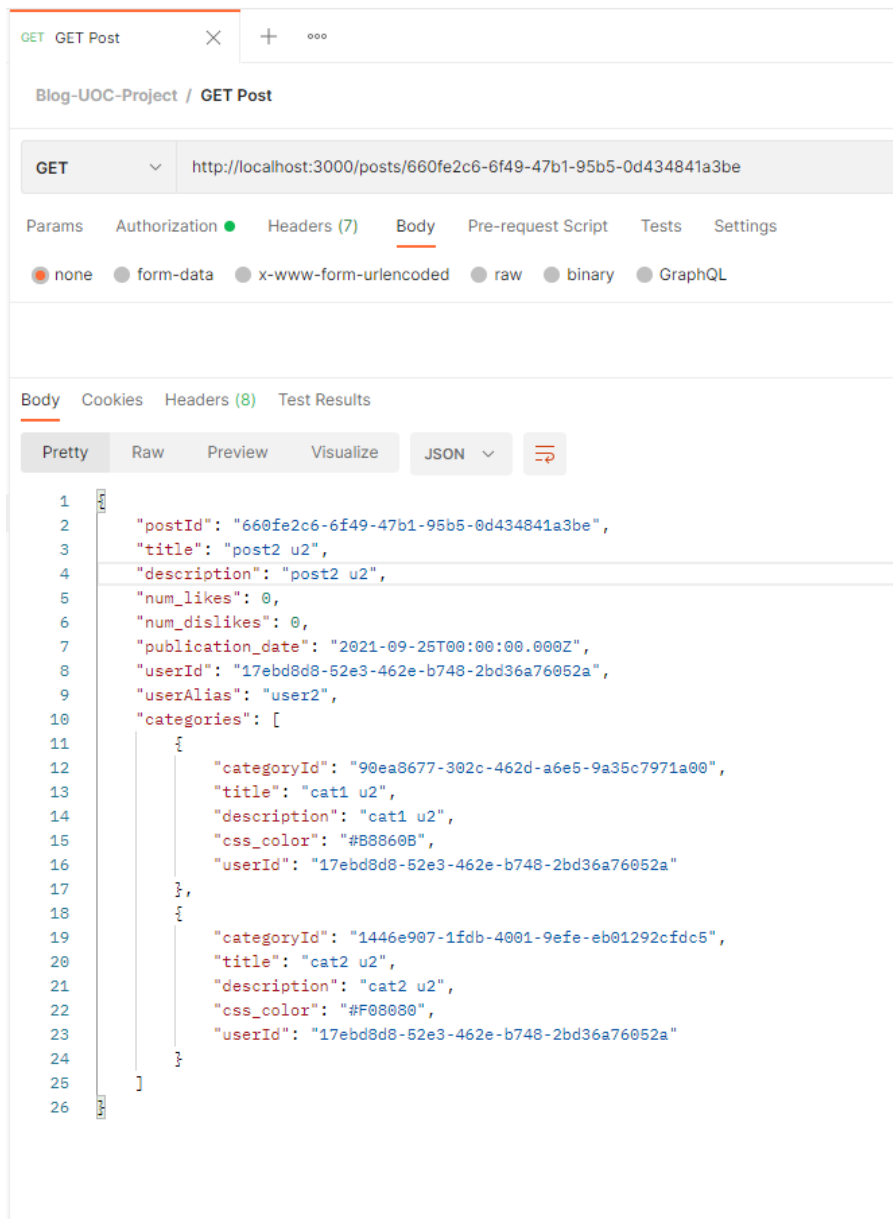
Pretty Raw Preview Visualize JSON

```

1  {
2    "postId": "4c8593c9-0cd1-445c-a892-27b313ff6b2c",
3    "title": "post1",
4    "description": "post1",
5    "num_likes": 5,
6    "num_dislikes": 6,
7    "publication_date": "2021-09-26T00:00:00.000Z",
8    "userId": "830203bb-6413-413b-a55f-75afdb5abea7",
9    "userAlias": "user1",
10   "categories": [
11     {
12       "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f956aff",
13       "title": "cat1",
14       "description": "cat1",
15       "css_color": "#F0F8FF",
16       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
17     },
18     {
19       "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
20       "title": "cat2",
21       "description": "cat2",
22       "css_color": "#FAEBD7",
23       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
24     }
25   ]
26 },
27 {
28   "postId": "86e04b4d-173f-4098-ae3f-06b7b8b69210",
29   "title": "post2",
30   "description": "post2",
31   "num_likes": 5,
32   "num_dislikes": 4,
33   "publication_date": "2021-09-26T00:00:00.000Z",
34   "userId": "830203bb-6413-413b-a55f-75afdb5abea7",
35   "userAlias": "user1",
36   "categories": [
37     {
38       "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f956aff",
39       "title": "cat1",
40       "description": "cat1",
41       "css_color": "#F0F8FF",
42       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
43     },
44     {
45       "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
46       "title": "cat2",
47       "description": "cat2",
48       "css_color": "#FAEBD7",
49       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
50     }
51   ]
52 }
53 ]
54

```

- Retornar un post segons el seu id:



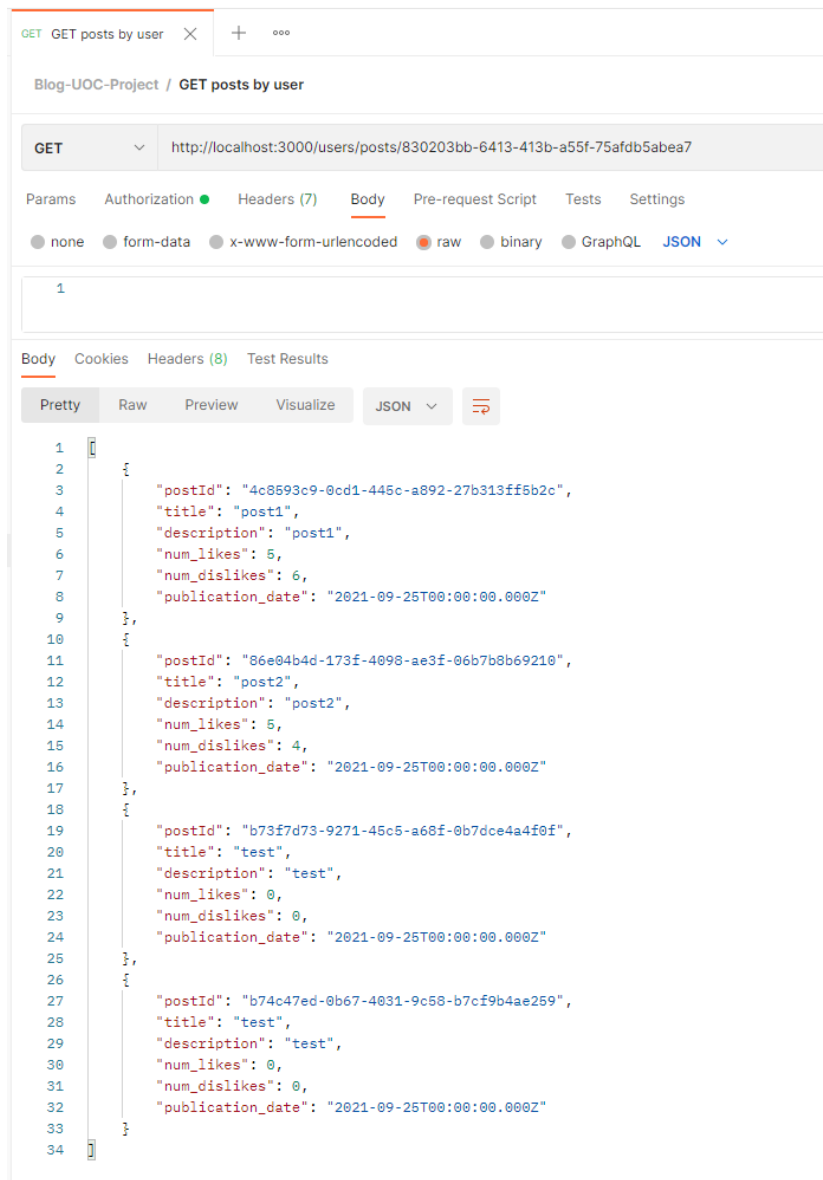
The screenshot shows a REST client interface with a GET request to `http://localhost:3000/posts/660fe2c6-6f49-47b1-95b5-0d434841a3be`. The response is a JSON object representing a post with various fields including ID, title, description, likes, dislikes, publication date, user ID, user alias, and a list of categories.

```

1  {
2    "postId": "660fe2c6-6f49-47b1-95b5-0d434841a3be",
3    "title": "post2 u2",
4    "description": "post2 u2",
5    "num_likes": 0,
6    "num_dislikes": 0,
7    "publication_date": "2021-09-25T00:00:00.000Z",
8    "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a",
9    "userAlias": "user2",
10   "categories": [
11     {
12       "categoryId": "90ea8677-302c-462d-a6e5-9a35c7971a00",
13       "title": "cat1 u2",
14       "description": "cat1 u2",
15       "css_color": "#B8860B",
16       "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a"
17     },
18     {
19       "categoryId": "1446e907-1fdb-4001-9efe-eb01292cfdc5",
20       "title": "cat2 u2",
21       "description": "cat2 u2",
22       "css_color": "#F08080",
23       "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a"
24     }
25   ]
26 }

```


- Retornar tots els posts d'un usuari concret:



GET GET posts by user

Blog-UOC-Project / GET posts by user

GET http://localhost:3000/users/posts/830203bb-6413-413b-a55f-75afdb5abea7

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies Headers (8) Test Results

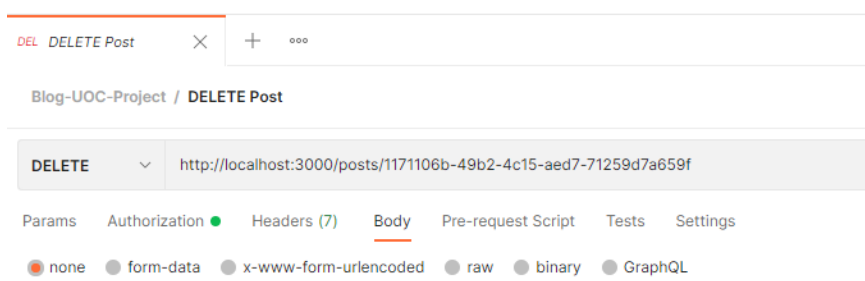
Pretty Raw Preview Visualize JSON

```

1  {
2    {
3      "postId": "4c8593c9-0cd1-445c-a892-27b313ff5b2c",
4      "title": "post1",
5      "description": "post1",
6      "num_likes": 5,
7      "num_dislikes": 6,
8      "publication_date": "2021-09-25T00:00:00.000Z"
9    },
10   {
11     "postId": "86e04b4d-173f-4098-ae3f-06b7b8b69210",
12     "title": "post2",
13     "description": "post2",
14     "num_likes": 5,
15     "num_dislikes": 4,
16     "publication_date": "2021-09-25T00:00:00.000Z"
17   },
18   {
19     "postId": "b73f7d73-9271-45c5-a68f-0b7dce4a4f0f",
20     "title": "test",
21     "description": "test",
22     "num_likes": 0,
23     "num_dislikes": 0,
24     "publication_date": "2021-09-25T00:00:00.000Z"
25   },
26   {
27     "postId": "b74c47ed-0b67-4031-9c58-b7cf9b4ae259",
28     "title": "test",
29     "description": "test",
30     "num_likes": 0,
31     "num_dislikes": 0,
32     "publication_date": "2021-09-25T00:00:00.000Z"
33   }
34 }

```

- Eliminar un post:



DEL DELETE Post

Blog-UOC-Project / DELETE Post

DELETE http://localhost:3000/posts/1171106b-49b2-4c15-aed7-71259d7a659f

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

- Retornar les categories d'un usuari:

GET GET categories by...
+

Blog-UOC-Project / GET categories by user

GET
http://localhost:3000/users/categories/830203bb-6413-413b-a55f-75afdb5abea7

Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings

Type
Bearer Token
Heads up! These parameters will be automatically generated when you send the request.
Learn more about authorization
Token

Body
Cookies
Headers (8)
Test Results

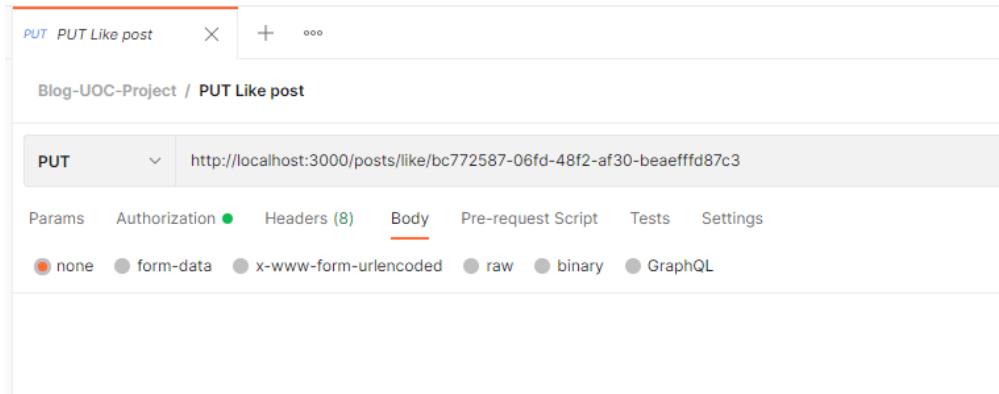
Pretty
Raw
Preview
Visualize
JSON

```

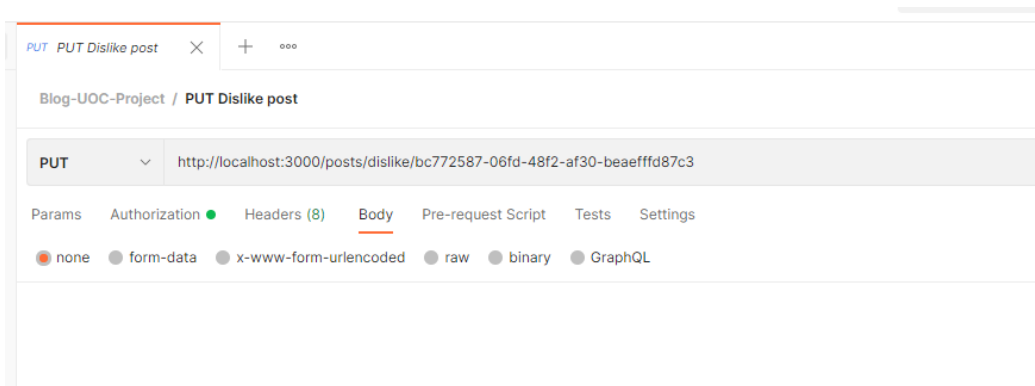
1  {
2    {
3      "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f955aff",
4      "title": "cat1",
5      "description": "cat1",
6      "css_color": "#F0F8FF"
7    },
8    {
9      "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
10     "title": "cat2",
11     "description": "cat2",
12     "css_color": "#FAEBD7"
13   },
14   {
15     "categoryId": "fa83e721-1009-4457-aadf-fdce1d0b456c",
16     "title": "cat3",
17     "description": "cat3",
18     "css_color": "#A52A2A"
19   }
20 }

```

- Enviar un like:



- Enviar un dislike:



Un cop tingueu configurat el **Postman**, us recomanem interactuar amb l'API.

Feu un petit joc de proves igual que us mostro al vídeo on us ensenyo el resultat del web, però des de **Postman**. D'aquesta manera, us acostumeu a utilitzar les crides utilitzant aquesta eina, els **body** que heu de enviar en algunes sol·licituds i, sobretot, les respostes que rebreu, les quals haureu de tractar al **frontend**.

Un cop fets els proves, podeu netejar la base de dades executant les següents instruccions **SQL**:

- `delete from public.categories;`
- `delete from public.posts;`
- `delete from public.users;`

Per executar sentències **SQL** al **pgAdmin**, podeu fer clic amb el botó dret del ratolí sobre el nom de la base de dades **blog-uoc-project** i seleccionar l'opció **Query Tool**. S'obrirà una pestanya on podeu escriure les vostres sentències **SQL**. És important tenir una certa habilitat en **SQL** a nivell bàsic.

Un cop us hàgiu familiaritzat amb tots els punts finals, seria el moment d'aixecar el **frontend** i implementar la pràctica pròpiament dita.

Només caldria baixar el fitxer del campus **blog-uoc-project-front.zip**, descomprimir-lo a la carpeta corresponent i executar els següents comandaments des de la consola:

- **npm install**
- **ng serve**

Enunciat

Quan comenceu amb el desenvolupament d'aquesta pràctica, hi ha dues qüestions importants a tenir en compte:

- D'una banda, intenteu resoldre els exercicis en l'ordre proposat, ja que la pràctica està dissenyada per ser progressiva.
- D'altra banda, reviseu tot el que teniu implementat, especialment:
 - Les rutes definides i les **guards** implementades. Fixeu-vos en la **guard** que valida si existeix el **access_token** al **localStorage**; si no existeix, redirigeix l'usuari a la pantalla **login**
 - Els diferents serveis implementats per a les diferents entitats: **user**, **category**, **post** y **auth**. En aquesta primera pràctica, utilitzem promeses. A la pràctica 2, a més d'aplicar **Redux**, modificarem les crides **http** a **observables**, de manera que tindrem les dues opcions estudiades.
 - **Serveis** auxiliars com:
 - **auth-interceptor**: que afegeix **access_token** als **headers** de totes les crides per poder utilitzar els **endpoints** protegits de l'API.
 - **header-menus**: que és interessant ja que mitjançant un **BehaviorSubject** podem gestionar els elements del menú quan estiguem autenticats o no. Un component envia les dades mitjançant el mètode **next**, i el component **header** en aquest cas utilitza el mètode **subscribe**. D'aquesta manera, es poden gestionar mitjançant variables booleanes quan mostrar els menús de la zona pública o privada.
 - **local-storage**: que simplement encapsula els mètodes d'accés al **localStorage**.
 - **Shared**: que gestiona el **toast** per proporcionar **feedback** sobre si la resposta de l'API és correcta o no.

Exercici 1

En el primer exercici, completarem la implementació del registre. Quan anem al punt de menú **Register** haurem d'inserir les nostres dades d'usuari i donar-nos d'alta a la plataforma. Per fer-ho, haurem de completar la implementació del component **register.component**, concretament:

Al controlador **register.component.ts**:

- Descomentar la declaració de variables

Cerqueu al projecte l'etiqueta:

TODO 16

- Al **constructor** haurem de:
 - Inicialitzar l'objecte **registerUser**.
 - Inicialitzar la variable **isValidForm** a **null**.
 - Inicialitzar les diferents propietats del formulari:
 - **name**: requerit, mínim 5 / màxim 25 caràcters.
 - **surname_1**: requerit, mínim 5 / màxim 25 caràcters.
 - **surname_2**: opcional, mínim 5 / màxim 25 caràcters.
 - **alias**: requerit, mínim 5 / màxim 25 caràcters.
 - **birth_date**: requerit, format 'yyyy-mm-dd.
 - **email**: requerit, format de **email** vàlid.
 - **password**: requerit, mínim 8 caràcters, màxim 16.
 - **registerForm**: definir el **formBuilder** amb totes les propietats anteriors.

Cerqueu al projecte l'etiqueta:

TODO 17

A la vista **register.component.html**:

- Implementar el formulari reactiu per a gestionar totes les propietats anteriors. El formulari al fer **submit** tindrà que cridar al mètode **register** que teniu implementat al controlador, només heu de descomentar el codi per a poder utilitzar-lo.

Cerqueu al projecte l'etiqueta:

TODO 18

- No elimineu el **div** de dalt de tot, vincula el **id="registerFeedback"** amb els estils del propi component i la gestió del **toast** quan es crida al **this.sharedService.managementToast(...)** per gestionar la resposta ok o ko de la API.

Exercici 2

En el segon exercici, completarem la implementació de l'autenticació al sistema. Quan anem al punt de menú **Login**, haurem de poder inserir les nostres credencials i autenticar-nos a la plataforma. Per fer-ho, haurem de completar la implementació del component **login.component**, concretament:

Al controlador **login.component.ts**:

- Descomentar la declaració de variables.

Cerqueu al projecte l'etiqueta:

TODO 19

- Al **constructor** haurem de:
 - Inicialitzar l'objecte **loginUser**.
 - Inicialitzar les diferents propietats del formulari:
 - email**: requerit, format de **email** vàlid.
 - password**: requerit, mínim 8 caràcters, màxim 16.
 - loginForm**: definir el **formBuilder** amb totes les propietats anteriors.

Cerqueu al projecte l'etiqueta:

TODO 20

A la vista **login.component.html**:

- Implementar el formulari reactiu per a gestionar totes les propietats anteriors. El formulari al fer **submit** tindrà que cridar al mètode **login** que teniu implementat al controlador, només heu de descomentar el codi per poder utilitzar-lo.

Cerqueu al projecte l'etiqueta:

TODO 21

- No elimineu el **div** de dalt de tot, vincula el **id="loginFeedback"** amb els estils del propi component i amb la gestió del **toast** quan es crida al **this.sharedService.managementToast(...)** per gestionar la resposta ok o ko de la API.

Exercici 3

En aquest exercici implementarem el **logout**.

Al fitxer **header.component.ts** al mètode **logout** hem d'implementar la següent lògica (**TODO 15**):

- Utilitzant el mètode **remove** del servei **localStorageService** eliminar el camp **user_id**
- Utilitzant el mètode **remove** del servei **localStorageService** eliminar el camp **access_token**
- Crear una constant **headerInfo** de tipus **HeaderMenus** amb les propietats:
 - **showAuthSection: false**
 - **showNoAuthSection: true**
- Afegir la següent línia per a actualitzar els punts de menú:

```
this.headerMenusService.headerManagement.next(headerInfo);
```

- I finalment redirigir a la **home**

Exercici 4

Quan estiguem autenticats al sistema i anem al punt de menú **Profile**, haurem de poder editar les nostres dades d'usuari. Per fer-ho, haurem de completar la implementació del component **profile.component**, concretament:

Al controlador **profile.component.ts**:

- Descomentar la declaració de variables.

Cerqueu al projecte l'etiqueta:

TODO 4

- Al **constructor** haurem de:
 - Inicialitzar l'objecte **profileUser**.
 - Inicialitzar la variable **isValidForm** a **null**.
 - Inicialitzar les diferents propietats del formulari:
 - **name**: requerit, mínim 5 / màxim 25 caràcters.
 - **surname_1**: requerit, mínim 5 / màxim 25 caràcters.
 - **surname_2**: opcional, mínim 5 / màxim 25 caràcters.
 - **alias**: requerit, mínim 5 / màxim 25 caràcters.
 - **birth_date**: requerit, format 'yyyy-mm-dd'.
 - **email**: requerit, formato de **email** vàlid.
 - **password**: requerit, mínim 8 caràcters, màxim 16.
 - **profileForm**: definir el **formBuilder** amb totes les propietats anteriors.

Cerqueu al projecte l'etiqueta:

TODO 5

A la vista **profile.component.html**:

- Implementar el formulari reactiu per a gestionar totes les propietats anteriors. El formulari al fer **submit** tindrà que cridar al mètode **updateUser** que teniu implementat al controlador, només heu de descomentar el codi per poder utilitzar-lo.

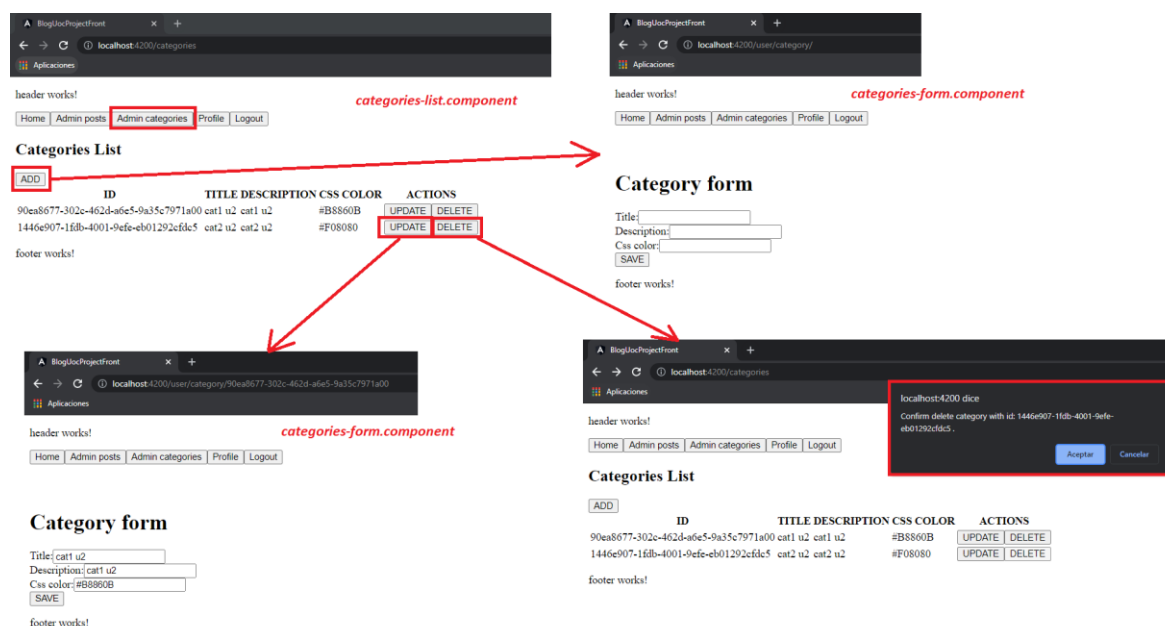
Cerqueu al projecte l'etiqueta:

TODO 6

- No elimineu el **div** de dalt de tot, vincula el **id="profileFeedback"** amb els estils del propi component i amb la gestió del **toast** quan es crida al **this.sharedService.managementToast(...)** per gestionar la resposta ok o ko de la API.

Exercici 5

En aquest exercici, acabarem d'implementar el CRUD de l'entitat de categories. Podeu tornar a veure el vídeo per veure com interactuar amb aquesta entitat. A mode de resum, podeu observar la següent imatge:



Proposem els següents passos:

- Estudia el component **categories-list**, tant la vista com el controlador.
- Observa com al controlador carreguem totes les categories en funció de l'usuari.
- Si fem clic al botó de crear una nova categoria redirigim al component **categories-form** sense passar un **categoryId**, afegeix la següent línia al mètode **createCategory** (**TODO 7**):

```
this.router.navigateByUri('/user/category/');
```

- Si fem clic al botó de editar una categoria redirigim al component **categories-form** passant un **categoryId** per **url**, afegeix la següent línia al mètode **updateCategory** (**TODO 8**):

```
this.router.navigateByUri('/user/category/' + categoryId);
```

- Si fem clic al botó d'eliminar una categoria, mostrarem un missatge de confirmació. Si pitgem **cancelar**, no es realitza cap acció. En canvi, si pitgem **acceptar**, s'eliminarà la categoria de la base de dades i haurem de recarregar la llista de categories de la pàgina actual. Per tant, afegim la següent línia al mètode **deleteCategory**(**TODO 9**):

```
this.loadCategories();
```

Valida el comportament anterior

- Anem a estudiar el component **categories-form**. En aquest component podem arribar tant si és un alta com si és una edició d'una categoria.
- Estudia el **constructor**, fixa't en com s'inicialitzen les diferents variables i els diferents camps del formulari.
- Estudia el **ngOnInit**, on gestionem si es tracta d'un alta o d'una edició mitjançant el camp **categoryId**. Recordem que l'enviem per **URL** si és una edició; si és un alta, no. En cas de ser una edició, sol·licitem les dades de la categoria en funció del seu **ID** per carregar-les al formulari.
- Estudia les dues funcions per editar i crear una categoria. Fixa't com llegim la identificació de l'usuari del **localStorage** i com fem la crida al servei per actualitzar o donar d'alta en funció del que es requereixi. Finalment, observa com utilitzem el servei **managementToast** per mostrar el feedback a l'usuari sobre si la resposta ha anat bé o no.
- Finalment, al mètode **saveCategory** ens falta implementar una petita lògica:
 - Substituir el **TODO 10** per:
 - Implementar un **if / else** de manera que:
 - Si la variable **isUpdateMode** es **true**:

```
this.validRequest = await this.editCategory();
```

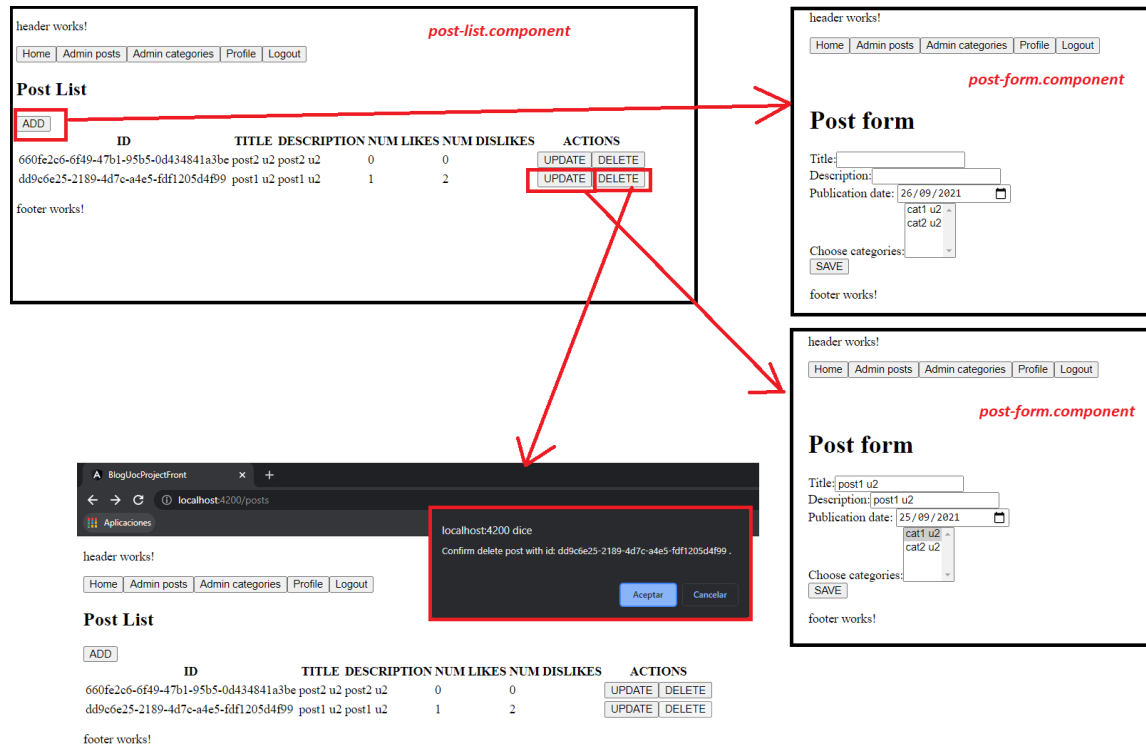
- Si no:

```
this.validRequest = await this.createCategory();
```

Exercici 6

En aquest exercici implementarem el CRUD de l'entitat d'entrades (posts). Aquest CRUD haureu de desenvolupar-lo completament des de zero. Per fer-ho, **podeu utilitzar com a referència el CRUD de categories anterior, ja que és molt similar en molts aspectes.**

El resultat serà el següent:



Al component **post-list.component** (**TODO 11** i **TODO 12**):

A la vista **post-list.component.html** haurem d'implementar:

- El botó d'afegir post.
- El llistat de posts amb un **ngFor** que mostri els camps:
 - postId.
 - title.
 - description.
 - num_likes.
 - num_dislikes.
 - Per cada fila les accions d'actualitzar i eliminar

Al controlador **post-list.component.ts** haurem d'implementar:

- Una variable **posts!**: **PostDTO[]**
- La càrrega de tots els posts de l'usuari mitjançant el servei **getPostsById**
 - o Haureu d'implementar el servei **getPostsById** del fitxer **post.service.ts** (**TODO 22**).
- El mètode **createPost** que redirigirà a **'user/post/'** sense passar un **postId**.
- El mètode **updatePost** que redirigirà a **'user/post/' + postId**.
- El mètode **deletePost** que demanarà confirmació per eliminar el post. Si es cancel·la la confirmació no es realitzarà cap acció i si es confirma es cridarà al servei **deletePost** i es tornarà a carregar el llistat de post per a que no aparegui l'últim eliminat.

Al component **post-form.component** (**TODO 13 y TODO 14**):

Al controlador **post-form.component.ts** haurem d'implementar tota la lògica al igual que al CRUD de categoria per diferenciar entre alta i edició i fer us del **managementToast** per mostrar **feedback** l'usuari.

- Respecte a les propietats del formulari de la entitat **post**:
 - o **title**: requerit, màxim 55 caràcters.
 - o **description**: requerit, màxim 255 caràcters.
 - o **publication_date**: requerit, format 'yyyy-mm-dd'.
 - o **categories**: array de categories, haurà de ser un **desplegable amb la opció de selecció múltiple**.
- Al controlador necessitem implementar:
 - o **loadCategories**: per a carregar les categories de l'usuari.
 - o **ngOnInit**: gestionarem el cas de si es una edició, per a carregar les dades del formulari.
 - o **editPost**: per a cridar el servei **updatePost** (haurem de d'aplicar la mateixa lògica que al CRUD de categories).
 - o **createPost**: per a cridar el servei **createPost** (haurem de d'aplicar la mateixa lògica que al CRUD de categories).
 - o **savePost**: si el formulari es vàlid en funció de si es alta o edició cridar a **editPost** o **createPost**.

A la vista **post-form.component.html** haurem d'implementar:

- Implementar el formulari reactiu per gestionar totes les propietats de l'entitat **post** (**title**, **description**, **publication_date** i **categories** associades). En fer **submit**, el formulari ha de cridar al mètode **savePost**.
- No elimineu el **div** de la part superior ja que vincula el **id="postFeedback"** amb els estils i amb el servei **managementToast**. Haureu de fer-ne ús de manera similar a com s'utilitza en el CRUD de categories.

Exercici 7

Al component **home.component**, tindrem la llista de totes les entrades (**posts**) de la plataforma. El component està per acabar. La idea seria mostrar la llista de les entrades més o menys com es mostra al vídeo. Podeu dissenyar l'aspecte com vulgueu. Algunes coses a tenir en compte:

- Al controlador **home.component.ts**: falta implementar la càrrega de tots els posts al mètode **loadPosts**. Per fer-ho, hem de seguir la lògica següent:

- Definir la variable **errorResponse**.
- Llegir del **localStorageService** el camp **userId**, si el tenim, assignar a la variable **showButtons** a **true**. Aquesta variable ens servirà després per mostrar o no els botons de "like" o "dislike" a la vista.
- Implementar el **try/catch** per cridar al **getPosts** i guardar el resultat a la variable **posts**.
- En cas d'error, passar-lo al **sharedService.errorLog**.

Cerqueu al projecte l'etiqueta:

TODO 2

- Implementació de la vista **home.component.html**: a la vista haurem d'implementar un **ngFor** de la variable **posts** i maquetar cada post. En cada post mostrarem tots els camps i a més totes les categories associades. Penseu que a cada categoria tenim que passar-li als estils el seu color de fons que ve definit per la propietat **category.css_color**. Finalment, en funció de la variable **showButtons** mostrarem els botones de **like** i **dislike**.

Cerqueu al projecte l'etiqueta:

TODO 3

Fixeu-vos com queda el resultat al vídeo explicatiu per tindre una referència.

Exercici 8

A la pantalla **home**, on tenim el llistat de tots els posts de la plataforma, de entre tots els camps d'un post tenim el camp data de publicació. Si mostrem el camp tal qual ens ve de base de dades veuríem una cosa així:



En aquest exercici es demana implementar un **pipe** de manera que doni un format amigable a la data. Concretament haurem d'implementar un **pipe** que rebí un numero com argument, de manera que:

- Passem un 1 i retorna el format → 25092021
- Passem un 2 i retorna el format → 25 / 09 / 2021
- Passem un 3 i retorna el format → 25/09/2021
- Passem un 4 i retorna el format → 2021-09-25

**** Penseu que si el número del dia o del mes es menor a 10 haurem de concatenar un zero al davant nosaltres de manera manual**

Cerqueu al projecte l'etiqueta:

TODO 1

Fixeu-vos com queda el resultat al vídeo explicatiu per tindre una referència.

Exercici 9

Un cop realitzada tota la implementació anterior i tenim la plataforma funcionant, anem a implementar un últim exercici obligatori. Els requisits són els següents:

- Implementar una nova ruta **dashboard** accessible des d'un punt de menú **DASHBOARD** de tipus públic, és a dir, no serà necessari estar autenticat al sistema per poder accedir a la vista.
- Implementar el component **dashboard** al qual navegarem des de la nova ruta anteriorment esmentada. En aquest component necessitem:
 - o Consumir l'**endpoint** que ens retorna totes les entrades (**posts**) de la plataforma (el mateix utilitzat en el component **home**).
 - o Al controlador del component **dashboard**, recórrer les entrades que ens retorna l'**endpoint** anterior i calcular el nombre total de **likes** i el nombre total de **dislikes**, i mostrar aquestes dues dades a la vista.
 - El càlcul s'haurà de fer al controlador, assignar-se a una variable pública i aquesta serà la que utilitzarem a la vista per mostrar les dues dades.

Exercici 10 (OPCIONAL)

La idea seria que una vegada acabada tota la implementació, si volem anar una mica més enllà, investiguem sobre aquestes dues propostes:

- D'una banda, a la vista **home**, on tenim la llista de tots els **posts** de la plataforma, podríem implementar algun tipus de filtre a la part superior de la llista. Per exemple, un filtre per data, un filtre per text que busqui el text si està inclòs en els títols o les descripcions dels posts, un filtre per alias, ... En definitiva, intentar aplicar algun filtre. Penseu que això seria a nivell de **frontend**, és a dir, amb la informació que teniu al **frontend** hauríeu de poder resoldre diversos exemples de filtres. No hi ha **endpoints** específics per a això.
- D'altra banda, un altre exercici que podríeu investigar és l'aplicació del paquet **i18n**, és a dir, fer la plataforma multi idioma. Bàsicament, la idea seria instal·lar el paquet i posar totes les traduccions en un fitxer **es.json** i **en.json** i substituir els textos **hardcode** de les vistes pels camps corresponents de **i18n**. Un cop configurat això, podríem implementar un desplegable a la part superior de l'aplicació on es pugui canviar entre els idiomes **es** i **en**.

Puntuació

A continuació, mostrem quan puntuen cada un dels exercicis de la pràctica. La pràctica puntua sobre 10 i tenim un punt extra si s'implementa l'exercici 10 que és opcional.

- Exercici 1 [**1 punt**]
- Exercici 2 [**1 punt**]
- Exercici 3 [**0,5 punts**]
- Exercici 4 [**1 punt**]
- Exercici 5 [**1 punt**]
- Exercici 6 [**3 punts**]
- Exercici 7 [**1 punt**]
- Exercici 8 [**1 punt**]
- Exercici 9 [**0,5 punt**]
- Exercici 10 [**1 punt**]